Exercise (10 minutes): process creation and synchronization

```c
#include <stdio.h>
#include <sys/wait.h> // wait()
#include <unistd.h> // fork()

void A() { printf("A completed \n"); }
void B() { printf("B completed \n"); }
void C() { printf("C completed \n"); }
void D() { printf("D completed \n"); }
void E() { printf("E completed \n"); }
```

```c
int main()
{
    int status;
    pid_t ...
    .
    .
    .
    return 0;
}
```

Hint: to wait for a certain process to complete, use

waitpid(pid, &status, 0);

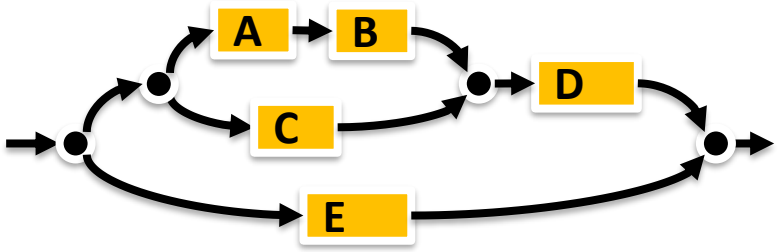
pid (should have a pid_t type) and is the ID of the process to wait for.

To fork a new process:

pid_t pid = fork();

If pid == 0, it is the child process, otherwise the parent

Write a pseudo-code to run functions A, B, C, D, and E on three concurrent processes such that the execution of these functions follows the diagram below (arrows show precedence constraints):



- C can run concurrently with A, B, and E but must be completed before D.
- B must be executed after A and before D.

- Build the solution together
- Use only three processes max

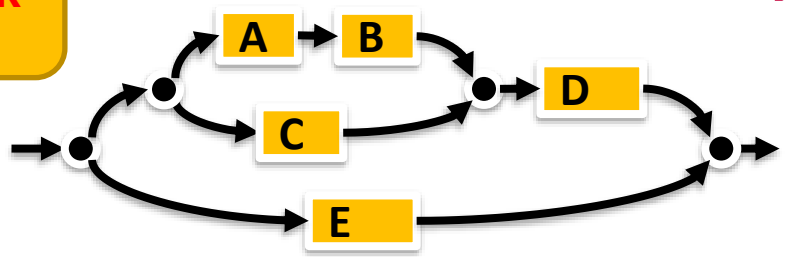
Exercise

!!! A fully correct solution would require to **check for error codes**

```
#include <stdio.h>
#include <sys/wait.h> // wait()
#include <unistd.h> // fork()

void A() { printf("A completed \n"); }
void B() { printf("B completed \n"); }
void C() { printf("C completed \n"); }
void D() { printf("D completed \n"); }
void E() { printf("E completed \n"); }

int main()
{
    int status;
    pid_t idC, idE;
    idE = fork();
    if (idE > 0)
    {
        idC = fork();
        if (idC > 0)
        {
           A();
           B();
           waitpid(idC, &status, 0); // for idC to join
           D();
           wait(NULL);
           printf("workload processed.\n");
        }
        else
        {   C();   }
     }
     else
     {   E();   }
    return 0;
}
```



- C can run concurrently with A, B, and E but must be completed before D.
- B must be executed after A and before D.